**Assignment #1**

**Software Testing**

**Submission Date: 21st Feb,2020 on slate**

|  |
| --- |
| import java.util.\*;  import java.lang.\*;  import java.io.\*;  class ShortestPath {  // A utility function to find the vertex with minimum distance value,  // from the set of vertices not yet included in shortest path tree  static final int V = 9;  int minDistance(int dist[], Boolean sptSet[])  {  // Initialize min value  int min = Integer.MAX\_VALUE, min\_index = -1;  for (int v = 0; v < V; v++)  if (sptSet[v] == false && dist[v] <= min) {  min = dist[v];  min\_index = v;  }  return min\_index;  }  // A utility function to print the constructed distance array  void printSolution(int dist[], int n)  {  System.out.println("Vertex Distance from Source");  for (int i = 0; i < V; i++)  System.out.println(i + " tt " + dist[i]);  }  // Function that implements Dijkstra's single source shortest path  // algorithm for a graph represented using adjacency matrix  // representation  int[] dijkstra(int graph[][], int src)  {  int dist[] = new int[V]; // The output array. dist[i] will hold  // the shortest distance from src to i  // sptSet[i] will true if vertex i is included in shortest  // path tree or shortest distance from src to i is finalized  Boolean sptSet[] = new Boolean[V];  // Initialize all distances as INFINITE and stpSet[] as false  for (int i = 0; i < V; i++) {  dist[i] = Integer.MAX\_VALUE;  sptSet[i] = false;  }  // Distance of source vertex from itself is always 0  dist[src] = 0;  // Find shortest path for all vertices  for (int count = 0; count < V - 1; count++) {  // Pick the minimum distance vertex from the set of vertices  // not yet processed. u is always equal to src in first  // iteration.  int u = minDistance(dist, sptSet);  // Mark the picked vertex as processed  sptSet[u] = true;  // Update dist value of the adjacent vertices of the  // picked vertex.  for (int v = 0; v < V; v++)  // Update dist[v] only if is not in sptSet, there is an  // edge from u to v, and total weight of path from src to  // v through u is smaller than current value of dist[v]  if (!sptSet[v] && graph[u][v] != 0 &&  dist[u] != Integer.MAX\_VALUE && dist[u] + graph[u][v] < dist[v])  dist[v] = dist[u] + graph[u][v];  }  // print the constructed distance array  printSolution(dist, V);  return dist;  }  // Driver method  public static void main(String[] args)  {  /\* Let us create the example graph discussed above \*/  int graph[][] = new int[][] { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },  { 4, 0, 8, 0, 0, 0, 0, 11, 0 },  { 0, 8, 0, 7, 0, 4, 0, 0, 2 },  { 0, 0, 7, 0, 9, 14, 0, 0, 0 },  { 0, 0, 0, 9, 0, 10, 0, 0, 0 },  { 0, 0, 4, 14, 10, 0, 2, 0, 0 },  { 0, 0, 0, 0, 0, 2, 0, 1, 6 },  { 8, 11, 0, 0, 0, 0, 1, 0, 7 },  { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };  ShortestPath t = new ShortestPath();  t.dijkstra(graph, 0);  }  } |

**For the above code:**

**Questions:**

a) Draw the control flow graph for each method.

b) Calculate cyclomatic complexity for each method.

c) Identify the paths

i) 100% Statement coverage ii) 100% branch coverage iii) Basis path coverage

d) Are there any infeasible paths in the code.

e) Based on the flow graph, prepare a minimum set of test cases for each coverage type. You may re-use test cases for the different coverage types.

f) Implement test cases using JUnit and the coverage tool EclEmma. If not already done, install the Code Coverage plug-in EclEmma (www.eclemma.org).